INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

DEPARTMENT OF ELECTRONICS, SYSTEMS, AND INFORMATICS

COMPUTING SYSTEMS ENGINEERING

MACHINE LEARNING COURSE

TEACHER: EDNA L. GUEVARA RIVERA

BIRD STRIKE FATALITY PREDICTION ON AIRPLANE CRASHES

PROJECT

PRESENTED BY:

MARCO RICARDO CORDERO HERNÁNDEZ, 727272

CARLOS EDUARDO RODRÍGUEZ CASTRO, 727366

DECEMBER 1ST, 2022

AUTUMN, 2022

TLAQUEPAQUE, MÉXICO

Index

Introduction 1
Problem to solve 2
Data collection 3
Learning type to use 4
Cleaning process introduction 5
Cleaning process walkthrough 6
Model implementation introduction 12
Model selection and motivation(s) 13
Implementation 14
    Linear Regression 15
    Neural Network 19
    Decision tree 26
Results description 29
Performance comparative 31
Conclusions and pending work 32
References 33

Introduction

It is often said that one is more likely to die in a car crash than in an airplane accident. This isn't an exaggeration, even being backed up by the United State National Safety Council [1]. This fact it's usually accompanied by the contrast of the highly likeliness of having an automobile accident in the way to an airport rather than in the plane itself. These aspects are backed up by the common knowledge of what is required in order to get a pilot's license versus the minimum aspects needed for a driver's license. Setting aside the economic resource needed for wings to fly, not anyone can become an airplane pilot, not even a private one, and those who do, they need to be in constant training [2]. For this sole reason, the probability of being in an aerial incident, fatal or not, it's very low. But, what about when there is indeed an accident? It can't be denied that human factor plays a big role in the final outcome of an aerial sinister, whether it's from land by air traffic controllers or by pilots stunned by unusual conditions [3]. Although it might seem contradictory to the first lines of this paragraph, the reality is that even by staying extremely calm, the most prepared and experienced cabin crew can't deal with a motor failure or complete loss in its entirety. This can be aided by analytics.

Machine learning (from now on referred as ML), as revised by Brown [4], may be seen as "the capability of a machine to imitate *intelligent human behavior*". Given this short but meaningful definition, the problem that this project will try to tackle can be seen as this: humans cannot think fast enough in a matter of life and death, whereas computers could certainly do.

By giving a proof by counterexample, Gupta [5] details two scenarios in which ML should be avoided thoughtfully: fairly ease or complexity lacking problems, and lack of labeled data. To put in someone's hands the life of several people it's not something to be taken lightly. The beautiful field of applied math conjoined with computer science usage could potentially save hundreds if not thousands of lives; just by applying simple algebra concepts such as matrices and dot products [6] great things can be achieved, solutions can be made and existing methods of avoiding fatalities can be drastically improved, in this case, through the application of ML. Although this it's just the introductory part of this work, it can be assured that poorly classified data or niche information won't be a problem in the becoming development.

Furthermore, and getting into a deeper level of detail, it's almost immediately recognized that the problem found can be addressed by applying supervised learning algorithms; as defined by Richards & Jia [7], these classifying algorithms make quantitative analysis over a dataset to decide whether an entry or set of entries correspond to some type of classification. This type of classification it's called like so because in order for it to work, desired outputs have to be given.

With the previous being said, it is not without reminding that ML it's just as strongest as its weakest link. ML it's a powerful tool, but it won't do miracles. In any case, the following sections will explore specific portions of the whole project.

Problem to solve

Ever since it happened, the US Airways flight 1549, or the "Miracle on the Hudson" as its often referred to, has become the flagship of aircraft incidents that turned out well in terms of fatalities. [8] [9] On January 15, 2009, said flight suffered a *bird strike* which led to a successful water landing, in which only injured passengers were reported, this meaning that no deaths were suffered on the incident. This is extremely rare, as the odds of surviving a plane crash versus those of an aquatic emergency landing are completely different [10]. At the moment of the incident, Chesley Sullenberger, the pilot that made the maneuvers for the successful landing, had over 40 years of experience or *training*, key factor in the fortunate outcome of the situation. With this in mind, does it really take a flight veteran to make or predict a favorable result in terms of lives lost?

Perhaps it might seem harmless at first glance, but when organic material such as birds' corpses get stuck into complex and carefully engineered machinery such as airplane turbines or helicopter rotors, disastrous events take place. The broken components of these aircrafts can be easily diagnosed with modern on-board systems, a detail of vastly interest, because with this piece of information, severity can be predicted ipso facto.

As a form of summarization, this project seeks to predict the fatality of a bird crash incident over type of aircraft, having such outcomes as *fatal* (0) and *non-fatal* (1).

Data collection

A dataset containing 25558 registers and 26 features has been retrieved from a data science platform [11].

The description of said set states that the values contained within the dataset comes directly from the Federal Aviation Administration (FAA), who provided the number and details of incidents where birds have struck a plane over a period of ten years, this being from 2000 to 2011 (two years after the Hudson incident).

With aid from pandas (a popular python data analysis library [12]), a quick analysis was made in order to determine the absence of values, which, in this case, was indeed found.

```
RangeIndex: 25558 entries, 0 to 25557
Data columns (total 26 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   record_id                              25558 non-null  int64
 1   aircraft_type                          25558 non-null  object
 2   airport_name                           25558 non-null  object
 3   altitude_bin                           25558 non-null  object
 4   aircraft_make_model                    25558 non-null  object
 5   wildlife_number_struck                 25558 non-null  object
 6   wildlife_number_struck_actual          25558 non-null  int64
 7   effect_impact_to_flight                25558 non-null  object
 8   flightdate                             25558 non-null  object
 9   effect_indicated_damage                25558 non-null  object
 10  aircraft_number_of_engines             25291 non-null  object
 11  aircraft_airline_operator              25558 non-null  object
 12  origin_state                           25109 non-null  object
 13  when_phase_of_flight                   25558 non-null  object
 14  conditions_precipitation               25558 non-null  object
 15  remains_of_wildlife_collected          25558 non-null  bool
 16  remains_of_wildlife_sent_to_smithsonian  25558 non-null  bool
 17  remarks                                20787 non-null  object
 18  wildlife_size                          25558 non-null  object
 19  conditions_sky                         25558 non-null  object
 20  wildlife_species                       25558 non-null  object
 21  pilot_warned_of_birds_or_wildlife      25558 non-null  bool
 22  cost_total                             25558 non-null  int64
 23  feet_above_ground                      25558 non-null  int64
 24  number_of_people_injured               25558 non-null  int64
 25  is_aircraft_large                      25558 non-null  bool
dtypes: bool(4), int64(5), object(17)
```

Fig. 1 Overview of dataset and analysis of values

The previous results show that the features number of engines, origin state and remarks not only have null values, but also that they all are objects, most likely strings.

Learning type to use

A supervised categorical algorithm has been chosen for this type of problem because there needs to be determined if a flight accident will or will not be fatal and our output will always be between "Fatal" and "Not fatal", ideally. In this case, the type of algorithm that'll be developed is categorical because it needs to classify all of the results under one of these two categories that are set.

One of the greatest advantages that this type of algorithm will bring to the project is that the result will be easily readable and no further processing is needed to extract real value from the output. Despite this, the algorithm has one disadvantage, debugging a categorical algorithm can be harder since there cannot be explicitly seen that an issue exists. The issue can only be detected when tests are made from the predicted results; since there is no complete control over specific operations the algorithm is doing, the debugging process can be quite time consuming.

When doing the comparison between the two main algorithm contenders (regression and categorical), discovers were made: even though regression can be considered a more precise algorithm, it lacks the output simplicity that the categorical algorithm is known for. All of the research points to use the categorical algorithm to predict whether a flight accident is fatal or not, the pros outweigh the cons for this specific application.

Cleaning process introduction

Through the last segments, the first stone has been set for the incoming steps that would encapsulate the knowledge gathered along the course for which this text has been written.

The most prevalent piece of work that needs to be made it's the transformation of the dataset as demonstrated in previous sections. Text or string fields were present, and, although this could be seem as problematic, the reality is that this information needs to undergo over a transformation and cleaning process in which these categorical data would be transformed into numerical values.

This process may vary depending on dataset structure, having multiple types of data crammed into several columns (referred as *features* from now on). For this particular project, additional measures had to be taken in order to transform string to numerical data.

The steps to clean the project's dataset are described below. Just as a reminder, the dataset comes from an external source [11].

Cleaning process walkthrough

First, libraries have to be imported in order to use their methods for data loading, manipulation, and visualization.

```python
# Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Next, the file containing the data itself has to be loaded. The file name it's *bird_strikes.csv*.

```python
# Read dataset
ds_bst = pd.read_csv('bird_strikes.csv')
```

After the previous step, the dataset can be visualized just by invoking the store valuable.

Relevant information related to data types for each feature has to be displayed to determine which columns could be kept.

```python
# Dataset info
ds_bst.info()
```

```
RangeIndex: 25558 entries, 0 to 25557
Data columns (total 26 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   record_id                              25558 non-null  int64
 1   aircraft_type                          25558 non-null  object
 2   airport_name                           25558 non-null  object
 3   altitude_bin                           25558 non-null  object
 4   aircraft_make_model                    25558 non-null  object
 5   wildlife_number_struck                 25558 non-null  object
 6   wildlife_number_struck_actual          25558 non-null  int64
 7   effect_impact_to_flight                25558 non-null  object
 8   flightdate                             25558 non-null  object
 9   effect_indicated_damage                25558 non-null  object
 10  aircraft_number_of_engines             25290 non-null  float64
 11  aircraft_airline_operator              25558 non-null  object
 12  origin_state                           25109 non-null  object
 13  when_phase_of_flight                   25558 non-null  object
 14  conditions_precipitation               25558 non-null  object
 15  remains_of_wildlife_collected          25558 non-null  bool
 16  remains_of_wildlife_sent_to_smithsonian 25558 non-null  bool
 17  remarks                                20787 non-null  object
 18  wildlife_size                          25558 non-null  object
 19  conditions_sky                         25558 non-null  object
...
 24  number_of_people_injured               25558 non-null  int64
 25  is_aircraft_large                      25558 non-null  bool
dtypes: bool(4), float64(1), int64(5), object(16)
```

Fig. 2 Dataset original features

Although output has been trimmed by the method containing library, critical information it's displayed at the bottom, indicating that 16 object type features (most likely strings) are present. Also, several boolean features are contained within other features, and although they could work in their original state, it's better to transform them into pure dichotomic values.

Before transforming present values, presence of null fields has to be taken into consideration.

```
# Null values identification
ds_bst.isna().sum(axis = 0)
```

```
record_id                                   0
aircraft_type                               0
airport_name                                0
altitude_bin                                0
aircraft_make_model                         0
wildlife_number_struck                      0
wildlife_number_struck_actual               0
effect_impact_to_flight                     0
flightdate                                  0
effect_indicated_damage                     0
aircraft_number_of_engines                268
aircraft_airline_operator                   0
origin_state                              449
when_phase_of_flight                        0
conditions_precipitation                    0
remains_of_wildlife_collected               0
remains_of_wildlife_sent_to_smithsonian     0
remarks                                  4771
wildlife_size                               0
conditions_sky                              0
wildlife_species                            0
pilot_warned_of_birds_or_wildlife           0
cost_total                                  0
feet_above_ground                           0
number_of_people_injured                    0
is_aircraft_large                           0
```

Fig. 3 Null presence in features

Through preliminary analysis, only one of the three null value containing features will have to be transformed into full data feature, this being *aircraft_number_of_engines*, as the other ones will be suppressed later on.

```
# Null values replacement
ds_bst['aircraft_number_of_engines'].fillna(value =
int(ds_bst['aircraft_number_of_engines'].mean()), inplace = True)

print(f"Null qty remaining: {ds_bst.isna().sum(axis =
0)['aircraft_number_of_engines']}")
```

```
Null qty remaining: 0
```

Fig. 4 Null absence verification

Just as the previous step it's completed, dropping the irrelevant features will take place. These droppable features are selected by looking at the information their data holds. Remaining features are shown below the step code.

```
# Non-relevant columns dropping
nrc = [
    'record_id',
    'airport_name',
    'wildlife_number_struck',
    'flightdate',
    'aircraft_airline_operator',
    'origin_state',
    'remains_of_wildlife_sent_to_smithsonian',
    'remarks',
    'wildlife_species',
    'cost_total'
]
'''
    Although 'cost_total' could be used, the information related to
that feature
    it's only obtained after the accident has occurred
'''

ds_bst.drop(nrc, inplace = True, axis = 1)
ds_bst.info()
```

```
Data columns (total 16 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   aircraft_type                    25558 non-null  object
 1   altitude_bin                     25558 non-null  object
 2   aircraft_make_model              25558 non-null  object
 3   wildlife_number_struck_actual    25558 non-null  int64
 4   effect_impact_to_flight          25558 non-null  object
 5   effect_indicated_damage          25558 non-null  object
 6   aircraft_number_of_engines       25558 non-null  float64
 7   when_phase_of_flight             25558 non-null  object
 8   conditions_precipitation         25558 non-null  object
 9   remains_of_wildlife_collected    25558 non-null  bool
 10  wildlife_size                    25558 non-null  object
 11  conditions_sky                   25558 non-null  object
 12  pilot_warned_of_birds_or_wildlife  25558 non-null  bool
 13  feet_above_ground                25558 non-null  int64
 14  number_of_people_injured         25558 non-null  int64
 15  is_aircraft_large                25558 non-null  bool
dtypes: bool(3), float64(1), int64(3), object(9)
```

Fig. 5 Remaining features after deletion

Now, just as stated before, object/string and bool features have to be transformed to numerical values. Two functions were made, one to transform categorical values, and another to turn boolean values to their binary representation.

```
def categorize(dataset, feature):
    holder = {}
    index = 0

    for row in dataset[feature]:
        if (row not in holder):
            holder[row] = index
            index += 1

    for val in holder:
        dataset[feature] = dataset[feature].replace([f'{val}'], holder[val])

def to_binary(dataset, feature):
    dataset[feature] = dataset[feature].apply(lambda x : 1 if x else 0)
```

With the aid of Fig. 4, indexes of each feature and their corresponding transformation can be done easily.

```
features = ds_bst.columns.values
to_modify = (0, 1, 2, 4, 5, 7, 8, 10, 11)
to_bin = (9, 12, 15)

# Implementation not recommended for long features lenght (<50)
for i in range(16):
    if (i in to_modify):
        categorize(ds_bst, features[i])
    elif (i in to_bin):
        to_binary(ds_bst, features[i])

ds_bst.info()
```

```
Data columns (total 16 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   aircraft_type                     25558 non-null  int64
 1   altitude_bin                      25558 non-null  int64
 2   aircraft_make_model               25558 non-null  int64
 3   wildlife_number_struck_actual     25558 non-null  int64
 4   effect_impact_to_flight           25558 non-null  int64
 5   effect_indicated_damage           25558 non-null  int64
 6   aircraft_number_of_engines        25558 non-null  float64
 7   when_phase_of_flight              25558 non-null  int64
 8   conditions_precipitation          25558 non-null  int64
 9   remains_of_wildlife_collected     25558 non-null  int64
 10  wildlife_size                     25558 non-null  int64
 11  conditions_sky                    25558 non-null  int64
 12  pilot_warned_of_birds_or_wildlife 25558 non-null  int64
 13  feet_above_ground                 25558 non-null  int64
 14  number_of_people_injured          25558 non-null  int64
 15  is_aircraft_large                 25558 non-null  int64
dtypes: float64(1), int64(15)
```

Fig. 6 Transformed remaining features

Heading towards end of cleaning process, a visualization approach has to be taken in order to detect repeated values.

```
ds_bst.hist(bins = 30, figsize = (20, 20), color = 'r')
```

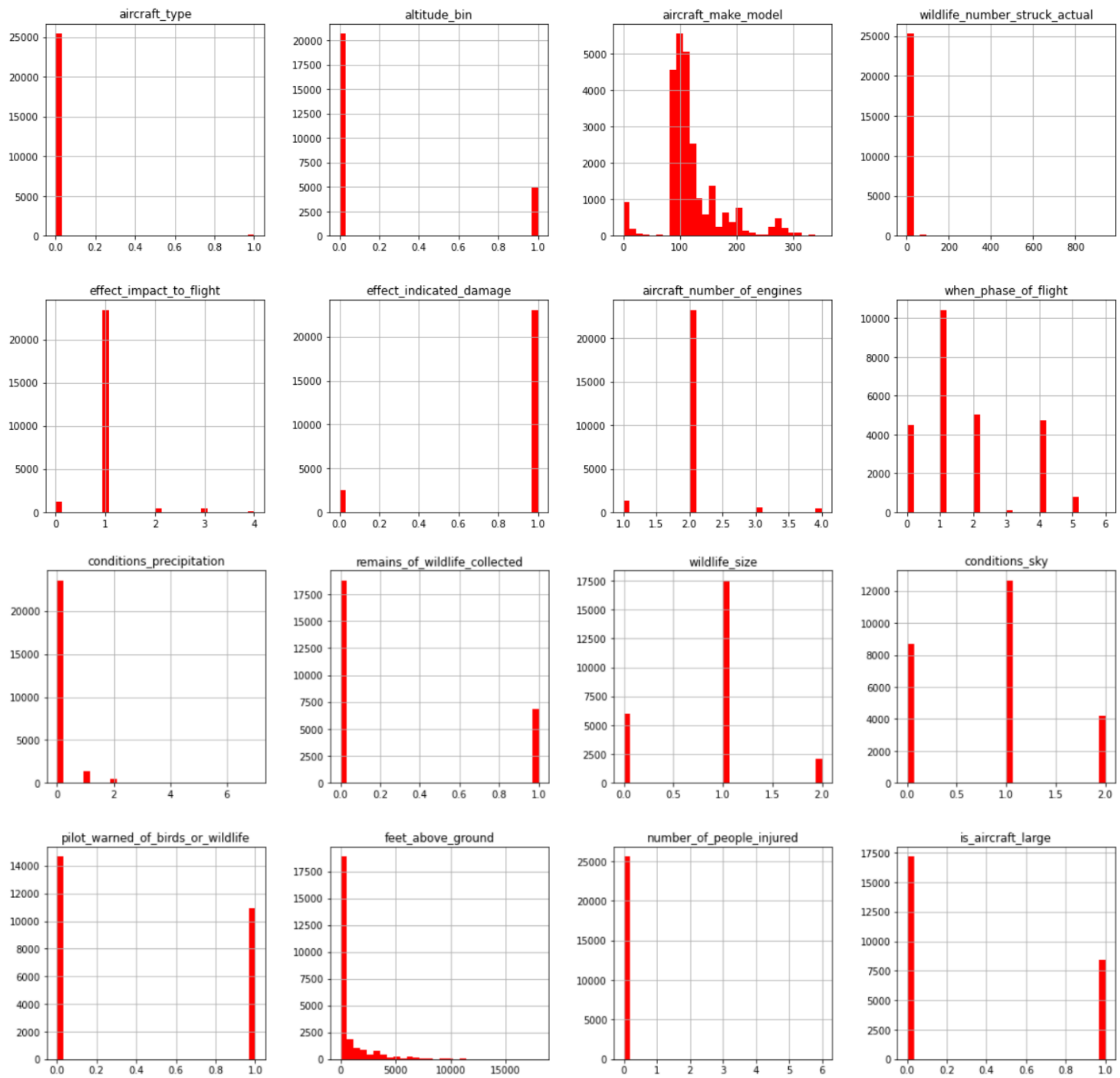


Fig. 7 Histograms

This extra analysis was helpful, because it ultimately helped in the exclusion of another two useless features.

```
ds_bst.drop(['aircraft_type', 'number_of_people_injured'], inplace =
True, axis = 1)
```

With that last step, the process of dataset cleaning has concluded. Additional analysis and clean dataset file generation can be found inside the jupyter notebook file.

Model implementation introduction

Considering the previous steps, the selected dataset has been cleaned through multiple programming and analysis techniques in order for it to be ready for machine learning algorithms, this with the purpose of training an effective and efficient set of models, from which one will be determined to be the best amongst them, at least for this problem.

As it has been used in past steps, Jupyter [13] will act as the container/holder for the computational operations results and outcomes from the algorithms.

Model selection and motivation(s)

For the analysis and comparison between results, and for class material comprehension purposes, the following Machine Learning Models will be implemented:

1. *Linear Regression (Normal)*
   As it is the most common type of technique and usually one of the first concepts used to teach about ML, this widely used model will function as the main comparison and example for further upgrading in next model implementations.
   Although the concepts involved in LR are fairly basic, these tools are still very useful and serve as a comparison entry point.

2. *Neural Network*
   Another broadly known technique when discussing about Machine/Deep Learning. This model has gained plenty of attention over recent years, as it's being used among a great range of modern-day problems, such as facial recognition, stock market predictions, signature verification, etc. [14] Thus, making it a great opportunity for a demonstration of this model for yet another contemporary problem.

3. Decision Tree
   Lastly, this model will be visited as an alternative to classic statistic methods, as DT's support nonlinear data and makes for a great visual resource that involves several categories/features found in the dataset of analysis. This highly customizable model allows for fine-grain knobbing/adjusting for better result outcome and can be easily compared against other models.

Implementation

Full model implementation can be found inside the jupyter notebook created specifically for this part of the project, giving an extensive explanation of its steps and result retrieving.

This file its embedded in the next figure (only accessible by Word; request original file at is727272@iteso.mx).

ML project_Part 3_ARPA.ipynb

First, and as per usual, libraries have to be loaded.

```
# Libraries
import math
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import tree
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

Then the cleaned dataset itself.

```
# Read dataset
ds_bst = pd.read_csv('bird_strikes_clean.csv')
```

	altitude_bin	aircraft_make_model	wildlife_number_struck_actual	effect_impact_to_flight	effect_indicated_damage	aircraft_number_of_engines	when_phase_of_flight
0	0	0	3	0	1	1.0	0
1	0	0	10	1	0	1.0	1
2	0	0	4	1	0	1.0	1
3	0	0	5	1	1	1.0	0
4	0	1	6	1	1	1.0	1
...	...	...	...	...	...	...	...
25553	0	313	1	1	0	2.0	1
25554	0	349	1	1	0	2.0	2
25555	0	248	1	0	1	2.0	0
25556	0	350	1	1	0	2.0	3
25557	0	248	1	1	0	2.0	0

25558 rows × 14 columns

Fig. 7 Trimmed dataset features

For ease of manipulation, feature swapping it's made by the following lines.

```
ds_bst['effect_indicated_damage'], ds_bst['is_aircraft_large'] =
ds_bst['is_aircraft_large'], ds_bst['effect_indicated_damage']
ds_bst.rename({'effect_indicated_damage': 'is_aircraft_large', 'is_aircraft_large':
'effect_indicated_damage'}, axis=1, inplace=True)
```

Linear Regression

```
Steps:
1. Convert dataset to numpy array
2. Add the columns of number 1
3. Split the dataset into Training and Testing sets
4. Using the xTrain and yTrain (Training dataset) and Linear Regression function
from sklearn library, obtain the model (W's). Then make predictions using the
Testing dataset, and obtain the R² score for predictions.
5. Using Ridge function from sklearn library, obtain the model (W's) and then make
predictions using the Testing dataset, and obtain the R² score for predictions.
6. Increment alpha value in logarithmic form: 10, 100, 1000, 10000, 100000, 1e6,
1e7, then graph ridge score behaviour for each alpha value
```

```
# Variable definition
ds_bst_np_lr = np.array(ds_bst)

x = ds_bst_np_lr[:, :-1]
y = ds_bst_np_lr[:, -1]
y = y.reshape(-1, 1)
```

```
# Add the columns of 1's
def addones(X):
    X1 = np.array(X)
    m, n = np.shape(X1)
    ones = np.ones((m, 1))
    X1 = np.concatenate((ones, X1), axis = 1)

    return X1

x = addones(x)
```

```
# Split the dataset into Training and Testing sets, test size of 33%,
and random_state= 1
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size =
0.33, random_state = 1)
print('Shape of Training data: ', np.shape(X_train), np.shape(y_train))
print('Shape of Testing data: ', np.shape(X_test), np.shape(y_test))
```

```
Shape of Training data:  (17123, 14) (17123, 1)
Shape of Testing data:  (8435, 14) (8435, 1)
```

Fig 8. Training and testing sub-datasets shapes

```
# Using the xTrain and yTrain (Training dataset) and Linear Regression function
from sklearn library, obtain the model (W's).
# Then make predictions using the Testing dataset, and obtain the  R2  score of
your predictions.
from sklearn.linear_model import LinearRegression

# Fit the data to training dataset
reg = LinearRegression().fit(X_train, y_train)

# Obtain and print the score
cost = reg.score(X_test, y_test)
print(f'Error (R2): {cost}')

# Obtain and print the W's coefficients
w = reg.coef_
print(f'W: {w}')


# Obtain and print the intercept
intercept = reg.intercept_
print(f'W0: {intercept}')


# Add the intercept value to the W's array and print W
w[0][0] = intercept
print(w)
```

```
Error (R2): 0.003701729051915792
W: [[ 0.00000000e+00  2.50363441e-02  1.59574603e-04  2.57883627e-04
   2.97686306e-02 -4.35096988e-02 -5.15405067e-02 -5.25606620e-03
   6.29354133e-03  3.38152096e-02  3.50695998e-02  1.18293780e-02
  -7.06727306e-03  8.31936216e-06]]
W0: [0.40910145]
[[ 4.09101451e-01  2.50363441e-02  1.59574603e-04  2.57883627e-04
   2.97686306e-02 -4.35096988e-02 -5.15405067e-02 -5.25606620e-03
   6.29354133e-03  3.38152096e-02  3.50695998e-02  1.18293780e-02
  -7.06727306e-03  8.31936216e-06]]
```

Fig. 9 Cost and weights

```
def r2(Y, Yt):
```

```
    error = Y - Yt
    variance = (Y - np.average(Y)) ** 2
    cost = 1 - (np.sum(error ** 2)) / np.sum(variance)
    return cost

# Predictions for Testing dataset
yt = np.dot(w, X_test.T).T
print(np.shape(yt))

# Obtain and print the R2 score
cost = r2(y_test, yt)
print(cost)
```

```
(8435, 1)
0.003701729051915792
```

Fig. 10 R^2 Output

```
# Linear regression con regularizacion "Ridge"
# Using Ridge function from sklearn library, obtain the model (W's) and then make
predictions using the Testing dataset
# and obtain the  R2  score of your predictions.
from sklearn.linear_model import Ridge

# Define the clf method using alpha = 10
clf = Ridge(alpha = 10.0)

# Fit to the training dataset
ridge = clf.fit(X_train, y_train)

# Obtain and print the score
Score2 = ridge.score(X_test, y_test)
print(f'R2: {Score2}')

# Obtain and print the W's coefficients
w2 = ridge.coef_
print(w2)

# Obtain and print the intercept
intercept2 = ridge.intercept_
print(intercept2)

# Add the intercept value to the W's array and print W
w2[0][0] = intercept2
print(w2)
```

```
R2: 0.0037144588612371132
[[ 0.00000000e+00  2.47830443e-02  1.57660449e-04  2.57857704e-04
   2.96454890e-02 -4.34310931e-02 -5.11185648e-02 -5.25358576e-03
   6.27735042e-03  3.37098567e-02  3.50118067e-02  1.18172227e-02
  -7.03214257e-03  8.35157199e-06]]
[0.40867754]
[[ 4.08677535e-01  2.47830443e-02  1.57660449e-04  2.57857704e-04
   2.96454890e-02 -4.34310931e-02 -5.11185648e-02 -5.25358576e-03
   6.27735042e-03  3.37098567e-02  3.50118067e-02  1.18172227e-02
  -7.03214257e-03  8.35157199e-06]]
```

Fig. 11 Costs and weights for ridge implementation

```
# Increment alpha value in logarithmic form: 10, 100, 1000, 10000, 100000,
1e6, 1e7
# then graph ridge score behaviour for each alpha value

alphas = [10, 100, 1000, 10000, 100000, 1e6, 1e7, 1e8]
J = []

for a in alphas:
    # Define the clf method using distinct alphas
    clf = Ridge(alpha = a)

    # Fit to the training dataset
    ridge = clf.fit(X_train, y_train)

    # Obtain and print the score
    Score = ridge.score(X_test, y_test)

    # Obtain and print the W's coefficients
    w = ridge.coef_

    # Obtain and print the intercept
    intercepto = ridge.intercept_

    # Add the intercept value to the W's array and print W
    w[0][0] = intercepto

    # Predictions for Testing dataset for Ridge algorithm
    yt = np.dot(w, X_test.T).T

    # Obtain and print the R2 score for Ridge Algorithm
    cost = r2(y_test, yt)

    J.append(cost)
```

```
plt.plot(alphas, J, 'b')
```

The generated graph will be displayed in results description section.

Neural Network

```
Steps:
1. Data loading
2. Plot the data
3. W function initialization, Sigmoid, Cost and Forward
4. Prediction, Accuracy and Decission Boundary
5. Model definition
6. Results visualization
```

```
# Plot the training dataset
f, ax = plt.subplots()
ax.plot(X_train, y_train)
plt.xlabel('X_test')
plt.ylabel('y_test')
plt.show()
```

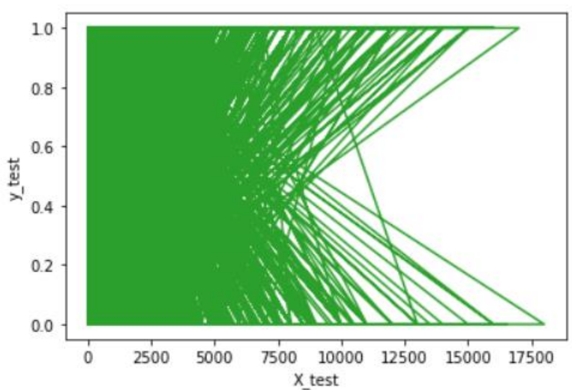


Fig. 12 Dataset plotting

```
# Initialize W's and b's
def init_w(m, nh, ny):
    np.random.seed(2)

    # w´s willbe created randomly
    # b's will be zeros
    W1 = np.random.randn(nh, m) * 0.01
```

```python
    b1 = np.zeros((1,nh))
    W2 = np.random.randn(nh, nh) * 0.01
    b2 = np.zeros((1,nh))

    W3 = np.random.randn(ny, nh) * 0.01
    b3= np.zeros((ny,1))
    W = {"W1": W1, "b1": b1, "W2": W2, "b2": b2, "W3": W3, "b3": b3}
    return W

# Testing the function
m = x.shape[1] # features on x
nh = 2 # hidden neurons
ny = 1 # outputs units

W = init_w(m, nh, ny)
print(W['W1'].shape, 'W1:\n', W['W1'])
print(W['b1'].shape, 'b1:\n', W['b1'])
print(W['W2'].shape, 'W2:\n', W['W2'])
print(W['b2'].shape, 'b2:\n', W['b2'])
print(W['W3'].shape, 'W3:\n', W['W3'])
print(W['b3'].shape, 'b3:\n', W['b3'])
```

```
(2, 13) W1:
 [[-4.16757847e-03 -5.62668272e-04 -2.13619610e-02  1.64027081e-02
  -1.79343559e-02 -8.41747366e-03  5.02881417e-03 -1.24528809e-02
  -1.05795222e-02 -9.09007615e-03  5.51454045e-03  2.29220801e-02
   4.15393930e-04]
 [-1.11792545e-02  5.39058321e-03 -5.96159700e-03 -1.91304965e-04
   1.17500122e-02 -7.47870949e-03  9.02525097e-05 -8.78107893e-03
  -1.56434170e-03  2.56570452e-03 -9.88779049e-03 -3.38821966e-03
  -2.36184031e-03]]
(1, 2) b1:
 [[0. 0.]]
(2, 2) W2:
 [[-0.00637655 -0.01187612]
 [-0.01421217 -0.00153495]]
(1, 2) b2:
 [[0. 0.]]
(1, 2) W3:
 [[-0.00269057  0.02231367]]
(1, 1) b3:
 [[0.]]
```

Fig. 12 Measures generated for the network

```python
# Sigmoid function
```

```python
def sigmoid(z):
    g = 1/(1+ np.exp(-z))
    return g

# Forward propagation to calculate ouput probabilites
def forward(x, W):
    W1 = W['W1']
    b1 = W['b1']
    W2 = W['W2']
    b2 = W['b2']

    W3 = W['W3']
    b3 = W['b3']

    a1 = x
    Z2 = np.dot(a1, W1.T) + b1

    a2 = sigmoid(Z2)
    Z3 = np.dot(a2, W2.T) + b2

    a3 = sigmoid(Z3)
    Z4 = np.dot(a3, W3.T) + b3

    a4 = sigmoid(Z4)
    Z = {'Z2': Z2, 'a2': a2, 'Z3': Z3, 'a3': a3, 'Z4': Z4, 'a4': a4}
    return a4, Z
```

```python
# Cost function
def cost(a, y):
    J = 1/2 * np.sum((a - y)**2)
    #J = np.sum((a - y)**2)
    return J

# Derivative of sigmoid function
def d_sigmoid(z):
    ds = sigmoid(z) * (1 - sigmoid(z))
    return ds
```

```python
# Backpropagation algorithm
def backp(W, Z, X, y):
    m = X.shape[1]

    W1 = W['W1']
```

```
    W2 = W['W2']
    W3 = W['W3']

    a2 = Z['a2']
    a3 = Z['a3']
    a4 = Z['a4']

    Z2 = Z['Z2']
    Z3 = Z['Z3']
    Z4 = Z['Z4']

    d4 = a4 - y
    d3 = np.dot(d4, W3) * d_sigmoid(Z3)
    d2 = np.dot(d3, W2) * d_sigmoid(Z2)

    dW1 = (1/m) * np.dot(d2.T, X)
    dW2 = (1/m) * np.dot(d3.T, a2)
    dW3 = (1/m) * np.dot(d4.T, a3)

    db1 = (1/m) * np.sum(d2, axis = 0)
    db2 = (1/m) * np.sum(d3, axis = 0)
    db3 = (1/m) * np.sum(d4)

    grad = {'dW1': dW1, 'dW2': dW2, 'dW3': dW3, 'db1': db1, 'db2': db2,
'db3': db3}
    return grad
```

```
# Implement and execute the NN model
def bird_strikes_model(x, y, nh, alpha = 0.001, epochs = 10000):
    np.random.seed(2)
    m = x.shape[1]
    ny = 1
    W = init_w(m, nh, ny)

    a4, z = forward(x, W)
    print('Initial cost:', cost(a4, y))

    J = []
    for i in range(epochs):
        a4, Z = forward(x, W)
        J.append(cost(a4, y))

        grad = backp(W, Z, x, y)
```

```
        W['W1'] = W['W1'] - alpha * grad['dW1']
        W['W2'] = W['W2'] - alpha * grad['dW2']
        W['W3'] = W['W3'] - alpha * grad['dW3']
        W['b1'] = W['b1'] - alpha * grad['db1']
        W['b2'] = W['b2'] - alpha * grad['db2']
        W['b3'] = W['b3'] - alpha * grad['db3']


    print('Final cost:', J[epochs-1])
    return W, J

W, J = bird_strikes_model(X_train, y_train, nh, alpha= 0.0001,
epochs=1000)

print('W1 =', W['W1'])
print("b1 = ", W['b1'])
print("W2 = ", W['W2'])
print("b2 = ", W['b2'])
print("W3 = ", W['W3'])
print("b3 = ", W['b3'])

plt.plot(J)
plt.title('Cost over epochs')
plt.xlabel('epochs')
plt.ylabel('cost');
```

```
Initial cost: 2241.0989198998973
Final cost: 2128.4205544298798
W1 = [[-0.00416695 -0.02541225 -0.02073006  0.01638013 -0.01815471 -0.00889685
   0.00459961 -0.01245113 -0.01053486 -0.00910445  0.00545763  0.02278818
   0.00966909]
 [-0.01117923  0.01496034 -0.0066245  -0.00019037  0.01184326 -0.00729661
   0.00027055 -0.00878733 -0.00158159  0.0025711  -0.00987532 -0.00333166
  -0.00631587]]
b1 =  [[-1.92683875e-04  7.75032958e-05]]
W2 =  [[-0.0354883   0.01787095]
 [-0.03939828  0.02314243]]
b2 =  [[0.00566973 0.00379399]]
W3 =  [[-0.1571611  -0.13472617]]
b3 =  [[-0.30040131]]
```

Fig. 13 Costs and measures for model implementation

The generated graph will be displayed in results description section.

```python
# Implement prediction, accuracy, and decision boundary functions
def predict(x, W):
    a4, Z = forward(x, W)
    y_hat = list(map(lambda x: 1 if x > 0.5 else 0, a4))
    y_hat = np.array(y_hat)
    y_hat = y_hat.reshape(-1, 1)
    return y_hat

def accuracy(y_hat, y):
    m = len(y)
    tptn = (y == y_hat).sum()
    acc = tptn / m
    return acc

def decision_boundary(x, y, w, ax):
    x_min, x_max = x[:, 0].min() - 0.5, x[:, 0].max() + 0.5
    y_min, y_max = x[:, 1].min() - 0.5, x[:, 1].max() + 0.5
    h = 0.01

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))

    Z1 = predict(np.c_[xx.ravel(), yy.ravel()], w)

    Z1 = Z1.reshape(xx.shape)

    ax.contourf(xx, yy, Z1, cmap =plt.cm.tab20c)
    ax.scatter(x[:, 0], x[:, 1], c = y.squeeze(), cmap=plt.cm.tab20c)
```

```python
# Prediction for Training dataset
y_hat = predict(X_train, W)
acc = accuracy(y_hat, y_train)
print(f'2 Neurons, accuracy = {str(acc)}')
```

```
2 Neurons, accuracy = 0.6097261039686976
```

Fig. 14 Model accuracy (Best outcome)

```
# Training with more neurons
hidden = [3, 4, 5, 6]

for h in hidden:
    W, J = bird_strikes_model(X_train, y_train, h, alpha= 0.0001,
epochs=1000)
    y_hat = predict(X_train, W)
    acc = accuracy(y_hat, y_train)

    print(f'{h} Neurons, accuracy = {acc}')
```

```
Initial cost: 2236.88541104206
Final cost: 2128.2322805695712
3 Neurons, accuracy = 0.6097261039686976
Initial cost: 2239.8152021854817
Final cost: 2128.3087753364944
4 Neurons, accuracy = 0.6097261039686976
Initial cost: 2238.2698948554125
Final cost: 2128.267501324638
5 Neurons, accuracy = 0.6097261039686976
Initial cost: 2233.027599373644
Final cost: 2128.084403253388
6 Neurons, accuracy = 0.6097261039686976
```

Fig. 15 Neuron addition costs and accuracy (training)

```
# Testing with same amount of neurons as training
y_hat = predict(X_test, W)
acc = accuracy(y_hat, y_test)
print(f'2 Neurons, accuracy = {acc}')

hidden = [3,4,5,6]

for h in hidden:
    W, J = bird_strikes_model(X_test, y_test, h, alpha= 0.0001,
epochs=1000)
    y_hat = predict(X_test, W)
    acc = accuracy(y_hat, y_test)
```

```
    print(f'{h} Neurons, accuracy = {acc}')
```

```
Initial cost: 2236.88541104206
Final cost: 2128.2322805695712
3 Neurons, accuracy = 0.6097261039686976
Initial cost: 2239.8152021854817
Final cost: 2128.3087753364944
4 Neurons, accuracy = 0.6097261039686976
Initial cost: 2238.2698948554125
Final cost: 2128.267501324638
5 Neurons, accuracy = 0.6097261039686976
Initial cost: 2233.027599373644
Final cost: 2128.084403253388
6 Neurons, accuracy = 0.6097261039686976
```

Fig. 16 Neuron addition costs and accuracy (testing)

Decision tree

```
Steps:
1. Data loading
2. Data analysis
3. Training and test separation
4. Gini and Entropy definition
5. Predictions
6. Tree plotting
7. Confusion matrix (both models)
8. Comparisons
```

```
X = ds_bst.values[:, :-1]
print(X)
Y = ds_bst.values[:, -1]

np.unique(Y, return_counts = True)
```

```
[[   0.    0.    3.  ...    0.    1. 1000.]
 [   0.    0.   10.  ...    1.    0.   20.]
 [   0.    0.    4.  ...    1.    0.  100.]
 ...
 [   0.  248.    1.  ...    1.    0.  800.]
 [   0.  350.    1.  ...    1.    0.    0.]
 [   0.  248.    1.  ...    1.    0.   50.]]

(array([0., 1.]), array([15573,  9985], dtype=int64))
```

Fig. 17 Dataset uniqueness verification

```
# Gini model
clf_gini = DecisionTreeClassifier(criterion = 'gini', random_state =
100, max_depth = 3, min_samples_leaf = 5)
clf_gini = clf_gini.fit(X_train, y_train)
```

```
# Entropy model
clf_entropy = DecisionTreeClassifier(criterion = 'entropy',
random_state = 100, max_depth = 3, min_samples_leaf = 5)
clf_entropy = clf_entropy.fit(X_train, y_train)
```

```
# Predictions
y_pred_gini = clf_gini.predict(X_test)
y_pred_entropy = clf_entropy.predict(X_test)

print(classification_report(y_test, y_pred_gini), '\n')
print(classification_report(y_test, y_pred_entropy))
```

```
              precision    recall  f1-score   support

         0.0       0.64      0.92      0.75      4665
         1.0       0.60      0.19      0.29      3003

    accuracy                           0.63      7668
   macro avg       0.62      0.55      0.52      7668
weighted avg       0.62      0.63      0.57      7668


              precision    recall  f1-score   support

         0.0       0.64      0.92      0.75      4665
         1.0       0.60      0.19      0.29      3003

    accuracy                           0.63      7668
   macro avg       0.62      0.55      0.52      7668
weighted avg       0.62      0.63      0.57      7668
```

Fig. 18 DT Predictions

```
# Tree ploting
plt.figure(figsize = (25, 10))
a = tree.plot_tree(clf_gini, filled = True, rounded = True, fontsize =
14)
```

```
plt.figure(figsize = (25, 10))
a = tree.plot_tree(clf_entropy, filled = True, rounded = True, fontsize
= 14)
```

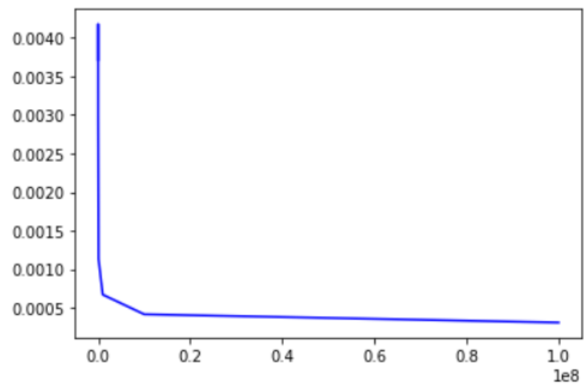
```
print('Train matrices')

cfm_train_gini = confusion_matrix(y_test, y_pred_gini)
print(cfm_train_gini, '\n')

cfm_train_entropy = confusion_matrix(y_test, y_pred_entropy)
print(cfm_train_entropy)
```

```
print('Test matrices')

# Gini model
clf_gini = DecisionTreeClassifier(criterion = 'gini', random_state =
100, max_depth = 3, min_samples_leaf = 5)
clf_gini = clf_gini.fit(X_test, y_test)

# Entropy model
clf_entropy = DecisionTreeClassifier(criterion = 'entropy',
random_state = 100, max_depth = 3, min_samples_leaf = 5)
clf_entropy = clf_entropy.fit(X_test, y_test)

# Predictions
y_pred_gini = clf_gini.predict(X_test)
y_pred_entropy = clf_entropy.predict(X_test)

cfm_train_gini = confusion_matrix(y_test, y_pred_gini)
print(cfm_train_gini, '\n')

cfm_train_entropy = confusion_matrix(y_test, y_pred_entropy)
print(cfm_train_entropy)
```

The generated graphs and matrix will be displayed in results description section.

Results description

Graph generated for Linear Regression

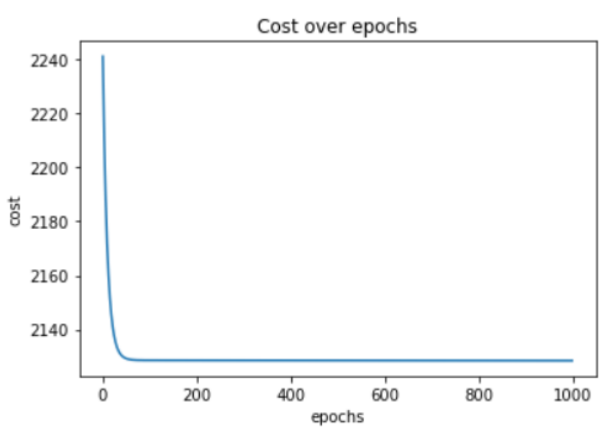



```
Error (R2): 0.003701729051915792
```

Fig. 19 Linear regression graph and error

For this model, great results have been achieved, as error is fairly low, and the graph demonstrates how cost descends elegantly, approaching zero.

Graph generated for Neural Network




```
2 Neurons, accuracy = 0.6083724569640062
Initial cost: 958.7690088398112
Final cost: 913.4016468898567
3 Neurons, accuracy = 0.6083724569640062
Initial cost: 960.0093254474551
Final cost: 913.4198081944725
4 Neurons, accuracy = 0.6083724569640062
Initial cost: 959.3550561132524
Final cost: 913.424485891052
5 Neurons, accuracy = 0.6083724569640062
Initial cost: 957.1368241622101
Final cost: 913.3861474025143
6 Neurons, accuracy = 0.6083724569640062
```

Fig. 20 Epoch graph and costs of neural network neuron addition

Yet again, costs descend as epochs augment, but this is usual behavior for NN. When analyzing cost, a fluctuation can be spotted, with similar costs repeating themselves, but with zero to no accuracy upgrades. This doesn't mean something has gone wrong or similar, but perhaps this could indicate that this model may not be adequate for this particular problem.

Tree graph and related data

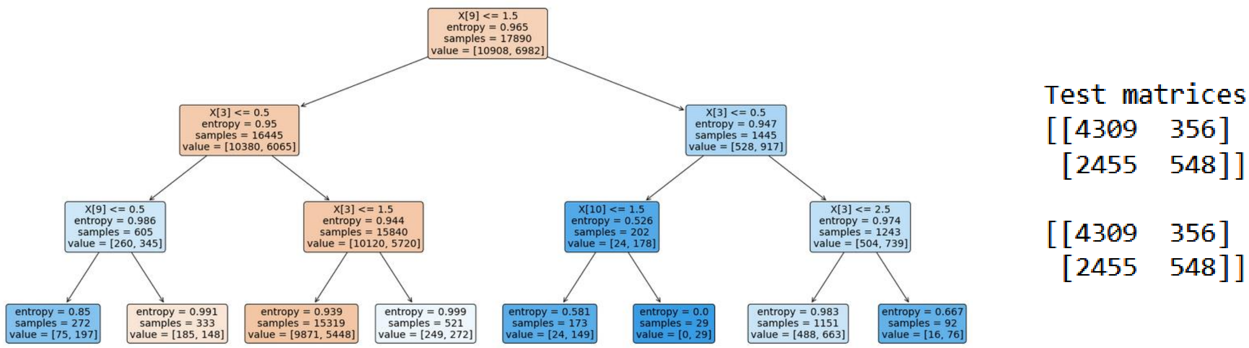


Fig. 21 Gini model prediction tree and matrices

Lastly, another common behavior can be spotted with this model execution when matrix analysis is made. Little under half the data it’s being correctly categorized, with about 60% of accuracy. It cannot be said that this model it’s bad or wrong, again, it’s just not the perfect fit for what it tried to address.

Performance comparative

Talking about time, the classifications remains as it follows (in ascending time order):

1. Linear Regression
2. Decision Tree
3. Neural Network

It has to be kept in mind that NN works closely with epoch concept and implementation, giving it the unfortunate last place, at least for performance.

Now, comparing results, it’s easy to place LR as the best model for the current problem, as it showed tiny error, meaning virtually no cost whatsoever, but something else can be said that would be more appropriate: further analysis has to be made. Whether is dataset refining, or data manipulation, or model selection, the results gathered in this part of the project can’t be totally seen as conclusive. But maybe that is what all of this it’s all about, about searching and building better and better models. The results obtained are not wrong, but maybe they would be serving a greater purpose as a simple entry point.

As for now, LR it’s the undeniable outstanding model. This’ll be discussed in the final document.

Conclusions and pending work

It can be considered now that the project has concluded, as the models for predictions has been implemented and its development has been demonstrated. Taking into account the level of detail required, the model implemented perhaps it's not the best suited for a new state of the art fatality outcome alert system, but, with some grain level refining, this can be achieved easily; in the end, the goal of the project was always aiming to improve modern aviation systems for future possible disasters involving mid-air strikes and/or collisions.

As an ambitious future goal, further data collection and manipulation could be done in order for a better model to be made, a model that could potentially save thousands of lives, endangered by air treacherous obstacles, such as birds and incompetence.

Talking about difficulties faced in project development, the utilized dataset had to be though in an abstract manner, thus being more than a collection of data, but rather a compendium of crammed information. Although it wasn't that much of a big deal, but the cleaning process was a cornerstone for the vast majority of model implementation, this being because the ultimate features selected to appear as critical and exclusively numerical data made possible the great results obtained, whereas leaving all information could harm the model(s) execution.

With this said, it can be asserted that probably it is not the model implementation what'll determine a successful outcome, but the cleaning process and data structure itself. Even the best of the models will execute poorly if random and uncorrelated values are used as input.

As for now, this project has concluded, but not before saying how much impact the knowledge gathered along its realization and the course that made it possible could potentially have in future. Today, the entry point for a collision fatality prediction system is set, tomorrow, never knows…

References

[1] National Safety Council, «Odds of Dying,» NSC Injury Facts, 2020. [Online]. Available: https://injuryfacts.nsc.org/all-injuries/preventable-death-overview/odds-of-dying/. [Last access: 14 September 2022].

[2] K. Hoke, «AeroSavvy,» AeroSavvy , 25 April 2018. [Online]. Available: https://aerosavvy.com/recurrent-training/. [Last access: 14 September 2022].

[3] Clifford Law Offices PC, «The National Law Review,» The National Law Review, 8 December 2020. [Online]. Available: https://www.natlawreview.com/article/most-common-causes-aviation-accidents. [Last access: 2022 September 14].

[4] S. Brown, «MIT Sloan School of Management,» MIT, 21 April 2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. [Last access: 14 September 2022].

[5] S. Gupta, «Springboard,» Springboard, 25 September 2020. [Online]. Available: https://www.springboard.com/blog/data-science/when-not-to-use-ml/. [Last access: 14 September 2022].

[6] Uniqtech, «Medium,» Data Science Bootcamp, 22 December 2018. [Online]. Available: https://medium.com/data-science-bootcamp/understand-dot-products-matrix-multiplications-usage-in-deep-learning-in-minutes-beginner-95edf2e66155. [Last access: 2022 September 2022].

[7] J. A. Richards y X. Jia, «Supervised Classification Techniques,» from *Remote Sensing Digital Image Analysis*, Berlin, Springe, 1999, pp. 181-222.

[8] C. Eastwood, Dirección, *Sully*. [Movie]. United States: Flashlight Films, 2016.

[9] S. Lanfermeijer, «Tailstrike,» Tailstrike Consultancy, [Online]. Available: 2022. [Last access: 15 September 2022].

[10] D. Null, «The Guardian,» 2011. [Online]. Available: https://www.theguardian.com/notesandqueries/query/0,5753,-10081,00.html. [Last access: 2015 September 2022].

[11] J. Shih, «data.world,» 2016. [Online]. Available: https://data.world/shihzy/2000-2011-birds-strikes-planes. [Last access: 15 September 2022].

[12] pandas, «pandas documentation,» 12 September 2022. [Online]. Available: https://pandas.pydata.org/docs/. [Last access: 15 September 2022].

[13] F. Pérez & B. Granger, «Jupyter project,» Jupyter project, 2022. [Online]. Available: https://jupyter.org/. [Last access: 14 November 2022].

[14] V. Kaushik, «Analytics Steps,» Analytics Steps Infomedia LLP, 26 August 2021. [Online]. Available: https://www.analyticssteps.com/blogs/8-applications-neural-networks. [Last acess: 15 November 2022].